Vol.2012-SLDM-155 No.2
Vol.2012-EMB-24 No.2
2012/3/2

組込みシステム向けメニーコア用 OpenCL 環境

稗田拓路[†1]　西山直樹[†1]　谷口一徹[†1]
冨山宏之[†1]　井上弘士[†2]

近年，大量のプロセッサコアを集積したメニーコアアーキテクチャが注目されている．並列動作可能なコア数が数十個〜数百あるいは数千のオーダに達するメニーコアアーキテクチャは，コアへのタスク割り当てを最適化することで，低消費電力と高い演算性能を満たす組込みシステムを実現することが可能なアーキテクチャであるが，タスク割り当てを柔軟に決定できる並列プログラミング環境が必要となる．本稿ではデータ並列実行ならびにタスク並列実行を実現する組込み向けメニーコアアーキテクチャを対象として，並列コンピューティング言語である OpenCL 環境の実装を示す．実験により，実装した OpenCL 環境がデータ並列実行ならびにタスク並列実行を実現できることを示す．

An OpenCL Environment with Many-core for Embedded Systems

Takuji Hieda,[†1] Naoki Nishiyama,[†1]
Ittetsu Taniguchi,[†1] Hiroyuki Tomiyama[†1]
and Koji Inoue[†2]

Many-core is regarded as attractive architecture in recent days. Typical many-core architecture has tens or hundreds of processing elements (PEs) which works concurrently. Many-core architecture can achieve low power consumption and highly computation performance for embedded systems to optimize task allocation on the PEs. However, parallel programming environment which can customize task allocation is required. This paper describes an OpenCL, a parallel computing language, environment implemented for many-core architecture which supports both data- and task-parallel programming models. Experimental results show that the implemented OpenCL environment achieves data- and task-parallel execution.

1. はじめに

半導体プロセスの微細化による動作速度の向上が限界に近づきつつあること，また消費電力や発熱の問題が顕著になっていることから，プロセッサコア単体の性能を向上させることが困難になっている．そのため，複数のプロセッサコアを並列駆動させることでシステムの処理能力を向上させるマルチコアアーキテクチャが主流となっている．特に組込みシステムにおいては，電池で駆動するシステムが多いことや熱対策が十分に施すことが困難であることなどから，システム全体を 1 つのチップ上に集積させた MPSoC (Multiprocessor System-on-Chip) が利用されている．

近年，マルチコアアーキテクチャよりさらに多くのコアを大量に集積したメニーコアアーキテクチャが注目を浴びている．メニーコアアーキテクチャの特徴として，演算ブロックの最小単位であるプロセッシングエレメント（PE）を大量に持つことで高い並列演算性能を持っているため，大量のデータを短時間で処理することが可能である．実用化されているメニーコアアーキテクチャの一つとして GPU (Graphics Processing Unit) が挙げられる．GPU は画像処理を目的として設計されており，PE 単体の処理能力は大きくないが，1 つの GPU あたり数百個の PE を持っている．グラフィックボードなどに搭載されており，一般に広く普及しているという利点があるため，GPU の高い並列演算能力を汎用演算に利用する GPGPU(General-purpose computing on GPUs) の研究が盛んに行われている．GPGPU を実現するための並列コンピューティング言語として，CUDA や OpenCL などが提案されている．しかし，GPU は元々画像処理専門のアーキテクチャであるため，同じ処理を大量のデータに対して適用するデータ並列実行は得意であるが，異なる複数の処理を並列実行するタスク並列を行うためには制約があり，また利用効率も悪くなる．このため，メニーコアアーキテクチャを効果的に使用するためには，実行するタスクへの PE 割り当てを最適化することが必要であるが，メニーコアアーキテクチャを有効利用するための研究はまだ発展途上にある．

本稿では，組込みシステムを想定したメニーコアアーキテクチャ上で動作する OpenCL 環境について述べる．提案システムは，データ並列実行ならびにタスク並列実行を実現する

†1 立命館大学理工学部
College of Science and Engineering, Ritsumeikan University
†2 九州大学大学院システム情報科学研究院
Department of Advanced Information Technology, Kyushu University

Vol.2012-SLDM-155 No.2
Vol.2012-EMB-24 No.2
2012/3/2

メニーコアアーキテクチャ向けに OpenCL 処理系を提供する．また，メニーコアアーキテクチャがハードウェアとして実装されていない場合，タスクをスレッド上で実行することで OpenCL プログラムを実行することが可能である．

本論文の構成を以下に述べる．まず 2 節で関連研究について述べる．次に 3 節で本研究で想定するメニーコアアーキテクチャについて説明した後，設計開発した OpenCL 処理系について述べる．処理系の有効性を確認するための評価実験ならびに結果について 4 節で示し，最後に 5 節でまとめる．

2. 関 連 研 究

メニーコアアーキテクチャは多くのコアを有するプロセッサであると定義されるが，マルチコアとメニーコアアーキテクチャの境界は明確には定まっていない．明確な区別をせずに同じように扱う場合もあれば，単純にコア数で区別する場合，あるいはコア単体の性能とコア数の相関関係で区別する場合などがある．現在では，コア単体の性能にもよるが，コア数が数十個以上であればメニーコアアーキテクチャであると言ってよいようである[1)]．メニーコアの代表的な利用例である GPGPU に関しては，これまでに様々な研究がなされている[2)]．GPGPU の研究が進むにつれて，GPU の並列処理能力を汎用処理に生かすための並列コンピューティング言語も複数提案されている．代表的な並列コンピューティング言語である CUDA は，nVidia 社が提供している自社製 GPU 向けの C 言語ベースの言語である[3)]．nVidia 社は後に述べる OpenCL の策定にも参加しており，CUDA と OpenCL には類似点が多くみられる．ただし，CUDA が nVidia 製 GPU のみを対象としているのに対し，OpenCL は様々なアーキテクチャを想定して設計されている違いがある．

OpenCL は Khronos グループが策定した並列コンピューティング用言語である[4)]．データ並列ならびにタスク並列プログラミングモデルをサポートしており，組込みシステムから HPC (High Performance Computing) まで幅広い用途に対応することができる．また，仕様が Khronos グループから公開されており，様々なプラットフォームへの移植が可能である．OpenCL では GPU などのメニーコアアーキテクチャを OpenCL デバイスとしてモデル化している．OpenCL デバイスのアーキテクチャモデルを図 2 に示す[4)]．OpenCL デバイスは複数の演算ユニットで構成され，それぞれの演算ユニットが PE を複数持つ．また，各演算ユニットごとにローカルメモリを，各 PE ごとにプレイベーとメモリをそれぞれ持つ．ローカルメモリは各演算ユニットごとに存在し，同じ演算ユニットに所属する PE からのみ読み書き可能である．また，プライベートメモリは各 PE ごとに存在し，プライベートメモリに接続されている PE からのみ読み書き可能である．また，ホストプロセッサからアクセスできる領域にグローバルメモリとコンスタントメモリが存在する．グローバルメモリはどの PE からも読み書き可能であるが，コンスタントメモリはどの PE からも読み込みのみ可能である．どちらのメモリについても，ホスト側からは読み書き可能であり，これらのメモリ領域を介してデバイス側とデータのやり取りを行う．

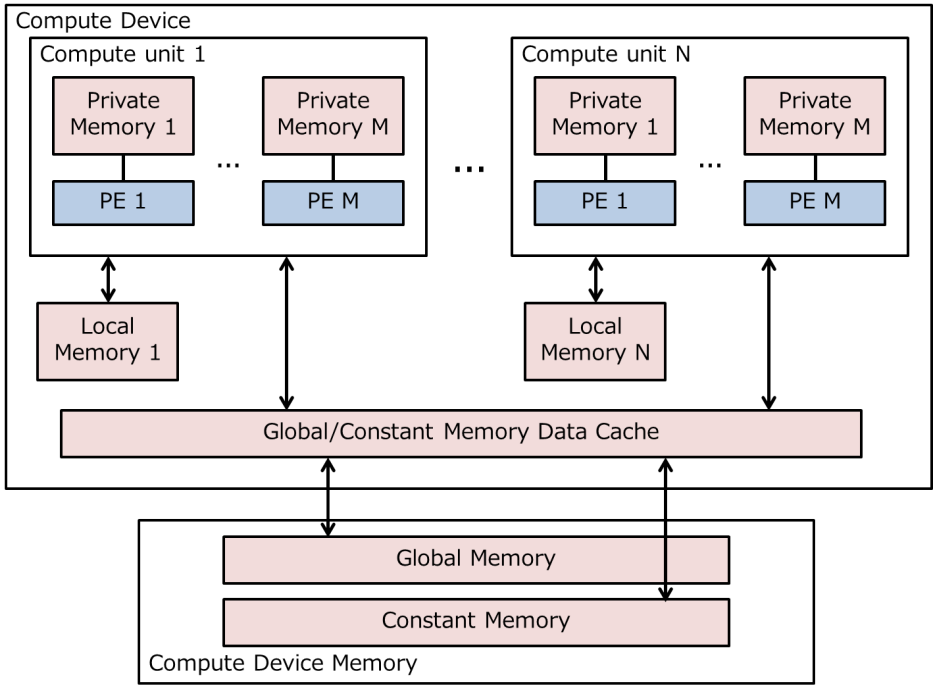

図 1 OpenCL デバイスアーキテクチャモデル
Fig. 1 OpenCL Device Architecture Model

現在，nVidia や ATI が GPU 上での実行を想定した OpenCL 環境を提供している．また，Intel Core プロセッサ向けの OpenCL 環境が提供されている．しかし，GPU は最小の処理単位が PE ではなく複数の PE をまとめたグループ単位であるため，1 つの PE だけを利用することができない[5)]．また Intel Core プロセッサ向け OpenCL では，現時点ではコア数が 10 個程度であるため，並列実行できるタスクの数がコア数に制限されることから，

Vol.2012-SLDM-155 No.2
Vol.2012-EMB-24 No.2
2012/3/2

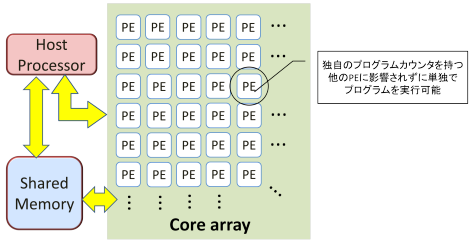

図 2 メニーコアアーキテクチャモデル図
Fig. 2 Many-core Architecture Model

メニーコア環境としては不完全な面が存在する．

3. OpenCL 処理系

本節では，柔軟なデータ／タスク並列処理をサポートしたメニーコア向けの OpenCL 環境について述べる．最初に想定するメニーコアアーキテクチャモデルについて説明し，その後で開発した OpenCL 処理系について述べる．

3.1 メニーコアアーキテクチャモデル

本稿で想定する組込みシステム向けメニーコアアーキテクチャモデルを図 2 に示す．想定するメニーコアアーキテクチャは，ホストプロセッサ部とコアアレイ部で構成されている．ホストプロセッサは，OS の実行や OpenCL API を用いて記述されたホスト側 C コードの実行などを担当する．コアアレイは複数の PE で構成されており，ホストプロセッサの指令に従って，OpenCL C 言語で記述されたカーネル関数の実行を担当する．ホストプロセッサとコアアレイ間でのデータのやり取りは共有メモリを介して行うことを想定している．コアアレイ部の特徴として，PE1 つ 1 つが専用のプログラムカウンタを持つため，それぞれ独立したプロセッサとしてカーネル関数を実行することが可能である．このため，コアブロック内では全て同じプログラムを動作させる必要がある GPU と違い，1 つ 1 つの PE ごとに違うプログラムを動作させることが可能である．この特性はタスク並列処理を実現する上で不可欠なものである．また，GPU と同様に複数の PE 上で同じプログラムを動作させることも可能である．これによりデータ並列処理を行うことが可能である．

図 2 で示したメニーコアアーキテクチャモデルでは，ホストプロセッサや PE に対する具体的なアーキテクチャは規定していないため，システム設計者が必要なアプリケーションに応じてホストプロセッサや PE の性能を決定することが可能である．ただし，ホストプロセッサ上で OpenCL プログラムを動作させるためには，ホストプロセッサ上で OpenCL ランタイムライブラリが動作することが必要となる．ランタイムライブラリは内部でスレッドライブラリを使用するため，スレッドライブラリが動作する環境であることがホストプロセッサ側に求められる．また，OpenCL では，penCL C 言語で規定されている組込み関数をカーネル関数内部で使用できる．これらの組込み関数は三角関数などの数学関数が中心であり，基本的な四則演算能力を有することが PE に求められる．また，PE の構成については，全て同じアーキテクチャの PE で統一したホモジーニアス構成にする必要はなく，一部の PE に特殊機能を持たせることができるが，これを OpenCL 処理系で利用する場合には処理系に特殊機能を利用できるようにするための追加実装が必要となる．

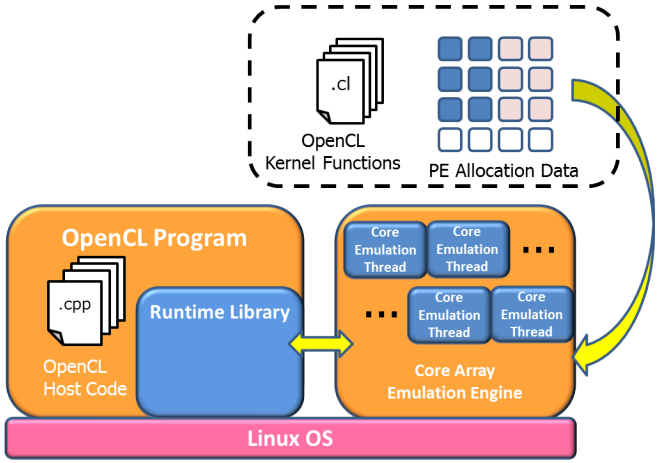

図 3 OpenCL 処理系概略図
Fig. 3 Overview of the OpenCL Implementation

3.2 メニーコアアーキテクチャ向け OpenCL 処理系

開発した OpenCL 処理系の概要を図 3 に示す．開発した処理系は Linux 環境下で動作する．現時点ではコアアレイ部はハードウェアとして存在せず，コアアレイ部の各 PE をそれぞれ 1 つのスレッドとしてエミュレーションすることでカーネル関数の実行を行う．そのため，本処理系は Linux が動作する環境であれば利用することが可能であり，アーキテクチャ非依存である．Linux は組込みシステムにおいても広く利用されているため，メニーコアを用いた組込みシステムを設計することが容易になる．また，エミュレーションを行う場

Vol.2012-SLDM-155 No.2
Vol.2012-EMB-24 No.2
2012/3/2

合はコア数を自由に設定可能である．

本処理系の特徴として，カーネル関数を実行する際のメニーコアへのタスクマッピングを制御可能である点が挙げられる．OpenCL で規定されている API では，カーネル関数を何個のタスクに分割するかを指定することが可能であるが，あるカーネル関数を実行する際に何個の PE を割り当てるか，ということは処理系の実装によって定められるため，アプリケーションプログラマが決定することはできない．そのため，複数のカーネル関数を並列実行したい場合であっても，あるカーネル関数がメニーコアの持つ PE 数よりも多くのタスクに分割された場合，全ての PE を使用して実行できるだけタスクを実行してから，残りのタスクを実行する．この間全ての PE が占有されるため，他のカーネル関数はタスクの終了を待たなければならなくなる．今回開発した OpenCL 処理系では，あらかじめ実行したいカーネル関数に対して何個のコアを割り当てるかを決めておくことで，それぞれのカーネル関数ごとに必要な分だけの PE を割り当てることで，カーネル関数の並列実行が可能とする．図 4 に既存のメニーコアアーキテクチャと比較した場合のコアアレイ部上での実行の様子を示す．左がタスク並列実行を行わない場合，右が行う場合である．どちらも 3 つのタスクを処理しており，それぞれのタスクが必要とする PE 数は左と右とで同数である．しかし，タスク並列実行ができない場合，使用されない PE が存在するため，コアアレイ部の使用効率が落ちている．そのため，左の場合にはコアアレイ部へのタスク割り当てを 7 回必要とするのに対して，右の場合は 5 回のタスク割り当てで実行が完了する．1 つのタスクだけ見ると利用できる PE 数が減っているためにタスクを完了するまでの実行時間は長くなるが，タスクの並列実行により全体的な実行効率を向上させることが可能である．

現時点では，OpenCL の仕様で規定されているホスト側 API のうち，以下の API が実装されている．

- `clCreateCommandQueue()` など，コマンドキュー作成に関する API
- `clEnqueueTask()`，`clSetKernelArg()` など，タスク実行に関する API
- `clCreateKernel()` など，カーネル関数作成に関する API
- `clEnqueueWriteMemory()` など，メモリ読み書きに関する API
- `clWaitForEvents()` など，イベントに関する API

これらの API はほぼ全ての OpenCL で使用される基本的な API であり，最低限必要な API である．また，これらの API を利用するために必要な OpenCL オブジェクトとして以下のオブジェクトが実装されている．

- コマンドキューオブジェクト: OpenCL デバイスへ処理を指示するためのコマンドキューを管理するオブジェクト
- カーネルオブジェクト: OpenCL デバイス上で実行するカーネル関数を管理するオブジェクト
- メモリオブジェクト: ホスト側とデバイス側とのデータやり取りに使用するメモリを管理するオブジェクト
- イベントオブジェクト: デバイスへの指示を実行時の状態やデバイス間の依存関係を管理するオブジェクト

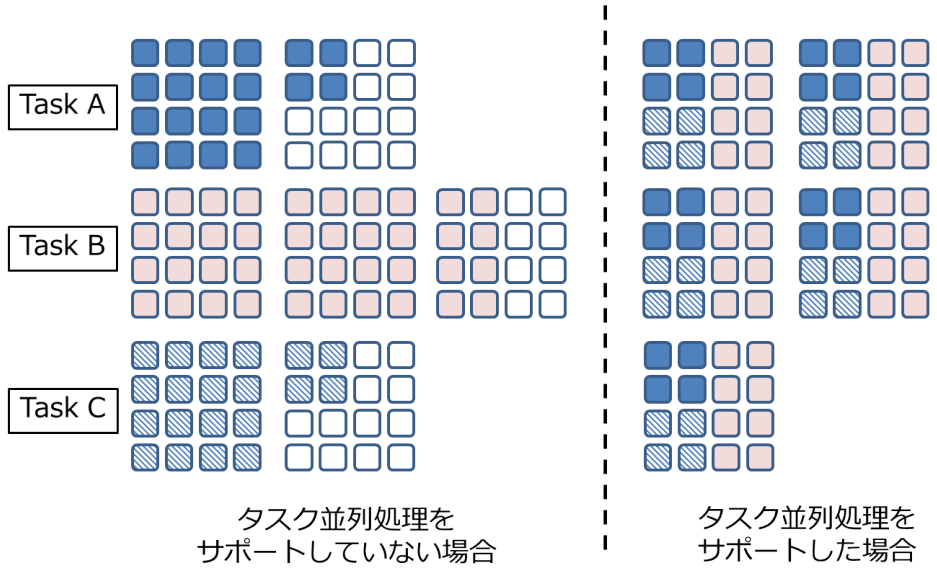

図 4 既存のメニーコアアーキテクチャとの比較
Fig. 4 Comparison with an Existing Many-core Architecture

画像データを専門に扱うイメージオブジェクトなど，必ずしも OpenCL プログラムに必要ない API に関しては現在実装されていないが，今後実装を進めていく方針である．また，現時点ではカーネル関数を記述するための OpenCL C 言語用コンパイラは実装されていない．このため，`clCreateProgramWithSource()` などのビルド用 API は未実装である．また，カーネル関数は C 言語の関数として実装し，ホスト側 OpenCL コードと共にコンパイル・リンクした状態で実行される．このため，カーネル関数内では基本データ型のみ使用可

Vol.2012-SLDM-155 No.2
Vol.2012-EMB-24 No.2
2012/3/2

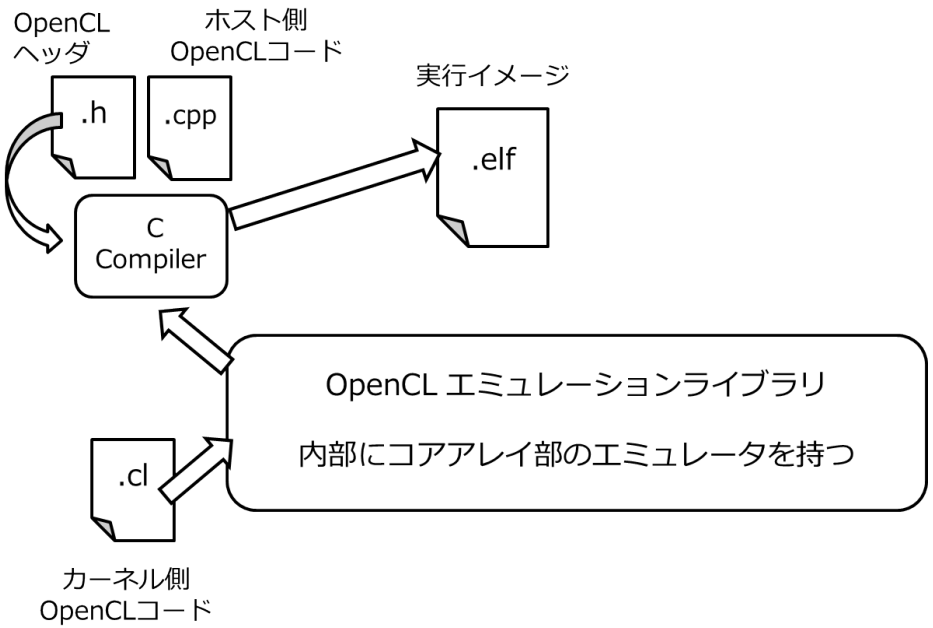

図 5 コアアレイ部をシミュレーションで実装する場合の OpenCL 処理系フロー
Fig. 5 The OpenCL Implementation Flow without Core-array Hardware

能であり，イメージ型やベクトル型については OpenCL C コンパイラの実装と共に今後実装していく予定である．

なお，コアアレイ部に関しては，FPGA 上への実装が現在行われており，完成次第 FPGA 上でカーネル関数が動作するように実装を行う予定である．コアアレイ部が FPGA を用いてハードウェア化された場合との違いを図 5 と図 6 に示す．図 5 において，エミュレーションを行う場合にエミュレーションライブラリが担当する部分を，図 6 ではコアアレイ部を実装したハードウェアを制御するデバイスドライバとランタイムライブラリが担当する．

4. 実　　験

前節で解説した OpenCL 処理系の有効性を確認するために，複数の OpenCL プログラムについて，開発した処理系と既存の OpenCL 処理系とでそれぞれ実行し，実行結果を比

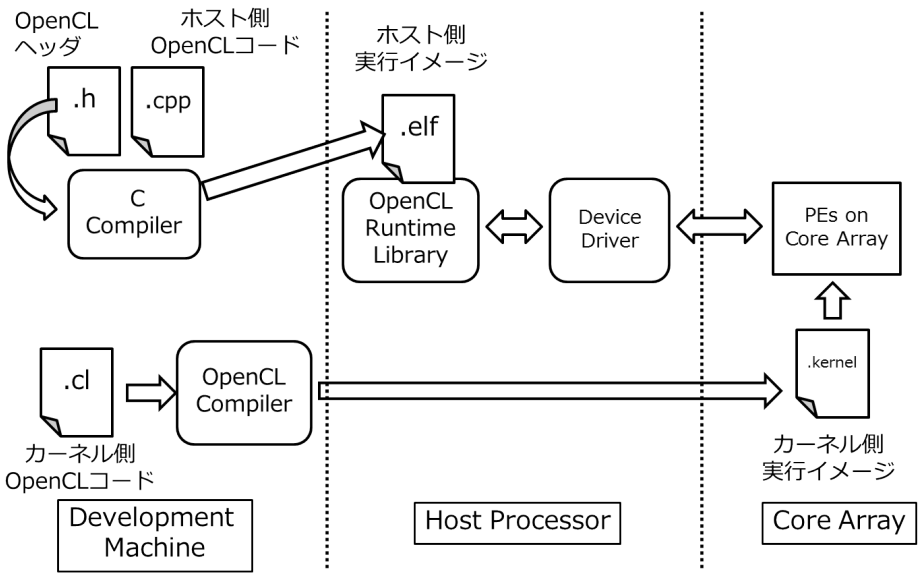

図 6 コアアレイ部がハードウェアとして実装されている場合の OpenCL 処理系フロー
Fig. 6 The OpenCL Implementation Flow with Core-array Hardware

較した．実験環境は以下の通りである．

- CPU: Core i7 860 2.8GHz
- Memory: DDR3-1333 8GB
- OS: OpenSUSE Linux 11.4

既存の OpenCL 処理系として，フィックスターズ社が提供する x86 向け OpenCL 環境である foxc[6] を利用した．テスト対象とした OpenCL プログラムは以下の 3 つである．

- `pipeline_test`：行列乗算と転置処理をパイプライン処理で実行するプログラム
- `linearsearch`：配列中のデータを線形検索するプログラム
- `blackScholes`：ブラックーショールズ方程式の計算を行うプログラム

`pipeline_test` は，組込みシステムでよく利用されるパイプライン処理をテストするためのプログラムである．行列の乗算処理を行うカーネル関数と，行列転置を行うカーネル関数の 2 つのカーネル関数で構成されており，行列を乗算する処理を行った後で，乗算結果の行列に対して行列転置を行う．2 つのカーネル関数は互いに独立しているため，乗算処理と転

Vol.2012-SLDM-155 No.2
Vol.2012-EMB-24 No.2
2012/3/2

表 1 各プログラムの実行時間
Table 1 Execution Time for each Program

	foxc [s]	Proposed OpenCL [s]
pipeline_test	0.37	0.22
linearsearch	13.7	2.3
blackScholes	1.90	0.03

置処理を並列実行することが可能である．これにより，パイプライン処理でプログラムを実行することが可能である．linearsearch は，1 次元配列に対して線形探索を行うプログラムである．この際，検索する範囲を分割することで，探索処理を並列実行する．分割数はタスク数と同じとする．blackScholes は，ブラック - ショールズ方程式の計算を行うプログラムである．複数の入力に対してカーネル関数を並列実行する．カーネル関数内で数学関数を使用しているため，カーネル側関数が動作するかの確認を行うことができる．

今回の実験では，仮想メニーコアアーキテクチャの PE 数を 64 として実験を行った．実験の結果，3 つの OpenCL プログラムともに正常動作することを確認した．Linux の time コマンドを使用して計測した実行時間を表 1 に示す．foxc を使用した場合は，今回開発した OpenCL 処理系に比べ実行時間が全体的に長くなっていることが分かる．理由としては，CPU を OpenCL デバイスとみなして問い合わせを行う処理が入るためと考えられる．

5. ま と め

本稿では，メニーコアアーキテクチャ向け OpenCL 環境を開発した．開発した OpenCL 環境は Linux 環境下で動作し，アーキテクチャに非依存で OpenCL 環境を動作させることが可能である．シミュレーションの結果，既存の OpenCL 環境と同等の動作を行わせることが可能であり，今後のメニーコアアーキテクチャの開発に役立てることが可能であるといえる．実際にメニーコアアーキテクチャをハードウェア上で構築し，OpenCL 環境として動作させることが課題として挙げられる．

謝辞 本研究は一部，独立行政法人新エネルギー・産業技術総合開発機構 (NEDO) の委託により実施した．本研究を実施するに当たり，有益なご助言を頂いた電気通信大学の近藤正章准教授,（株）フィックスターズ,（株）トプスシステムズの諸氏に感謝する．

参 考 文 献

1) Borkar, S.: Thousand core chips: a technology perspective, *Proceedings of the 44th annual Design Automation Conference*, DAC '07, New York, NY, USA, ACM, pp. 746–749 (2007).
2) Owens, J.D., Luebke, D., Govindaraju, N., Harris, M., Krüger, J., Lefohn, A. and Purcell, T.J.: A Survey of General-Purpose Computation on Graphics Hardware, *Computer Graphics Forum*, Vol.26, No.1, pp.80–113 (2007).
3) NVIDIA Corporation: *NVIDIA CUDA C Programming Guide, version 4.0*, available from ⟨http://developer.download.nvidia.com/compute/cuda/4_0/toolkit/docs/CUDA_C_Programming_Guide.pdf⟩ (2011).
4) Khronos OpenCL Working Group: *The OpenCL Specification Version 1.1*, available from ⟨http://www.khronos.org/registry/cl/specs/opencl-1.1.pdf⟩ (2011).
5) Lindholm, E., Nickolls, J., Oberman, S. and Montrym, J.: NVIDIA Tesla: A Unified Graphics and Computing Architecture, *IEEE Micro*, Vol.28, pp.39–55 (2008).
6) Fixstars Corporation: *FOXC (Fixstars OpenCL Cross Compiler)*, available from ⟨http://www.fixstars.com/ja/foxc/⟩ (2009).